

1. C Programming

No Deliverables

2. Displaying and Exporting Images in Python

No Deliverables

3. FIR Low Pass Filter

1. Derivation of $H(e^{j\mu}, e^{j\nu})$: *See Next Page*

$$h(m, n) = \begin{cases} 1/81 & \text{for } |m| \leq 4 \text{ and } |n| \leq 4 \\ 0 & \text{otherwise} \end{cases}$$

Identity Used:

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$

Jack Girard
ECE637
Lab 1 Work

Apply formula for DSFT & Simplify:

$$H(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m, n) e^{-j(\mu m + \nu n)}$$

$$= \sum_{n=-4}^4 \sum_{m=-4}^4 \frac{1}{81} e^{-j(\mu m + \nu n)}$$

$$= \frac{1}{81} \sum_{n=-4}^4 e^{-j\nu n} \sum_{m=-4}^4 e^{-j\mu m}$$

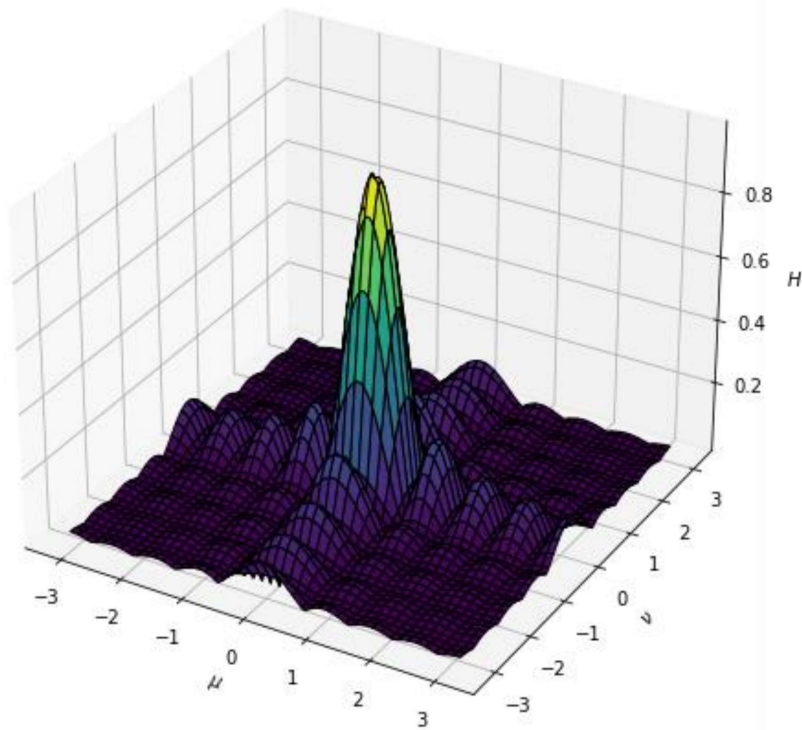
$$= \frac{1}{81} \left[(1 + e^{-4j\mu} + e^{4j\mu} + e^{-3j\mu} + e^{3j\mu} + e^{-2j\mu} + e^{2j\mu} + e^{-j\mu} + e^{j\mu}) \dots \right. \\ \left. (1 + e^{-4j\nu} + e^{4j\nu} + e^{-3j\nu} + e^{3j\nu} + e^{-2j\nu} + e^{2j\nu} + e^{-j\nu} + e^{j\nu}) \right]$$

$$= \frac{1}{81} \left[(1 + 2\cos(4\mu) + 2\cos(3\mu) + 2\cos(2\mu) + 2\cos(\mu)) \dots \right. \\ \left. (1 + 2\cos(4\nu) + 2\cos(3\nu) + 2\cos(2\nu) + 2\cos(\nu)) \right]$$

$$= \frac{1}{81} \left[\left(1 + 2 \sum_{k=1}^4 \cos(k\mu)\right) \left(1 + 2 \sum_{l=1}^4 \cos(l\nu)\right) \right]$$

2. Plot of $|H(e^{j\mu}, e^{j\nu})|$

Magnitude of Frequency Response $|H(e^{j\mu}, e^{j\nu})|$ vs μ and ν



3. Color image *img03.tif*



4. The filtered image *color.tif*



5. Listing of C Code: *See Next Page*

```
1
2 #include <math.h>
3 #include "tiff.h"
4 #include "allocate.h"
5 #include "randlib.h"
6 #include "typeutil.h"
7
8 void error(char *name);
9 // initialize limitIntensity function
10 int limitIntensity(double value);
11 // initialize applyFilter function
12 void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, ↗
    double** usedFilter, int PSF_dim);
13
14 int main (int argc, char **argv)
15 {
16     FILE* fp;
17     struct TIFF_img input_img, color_img;
18
19     if ( argc != 2 ) error( argv[0] );
20
21     /* open image file */
22     if ( ( fp = fopen(argv[1], "rb") ) == NULL ) {
23         fprintf( stderr, "cannot open file %s\n", argv[1] );
24         exit( 1 );
25     }
26
27     /* read image */
28     if ( read_TIFF( fp, &input_img ) ) {
29         fprintf( stderr, "error reading file %s\n", argv[1] );
30         exit( 1 );
31     }
32
33     /* close image file */
34     fclose( fp );
35
36     /* check the type of image data */
37     if ( input_img.TIFF_type != 'c' ) {
38         fprintf( stderr, "error: image must be 24-bit color\n" );
39         exit( 1 );
40     }
41
42     // declare 1D array of double pointers filter
43     double* filter[9];
44
45     // iterate through 1D array filter and allocate enough memory for 9 ↗
    doubles each
46     for (int i = 0; i < 9; i++) {
47         filter[i] = malloc(sizeof(double) * 9);
```

```
48     }
49     // populate filter array according to h(m,n)
50     for (int i = 0; i < 9; i++) {
51         for (int j = 0; j < 9; j++) {
52             filter[i][j] = 1.0 / 81.0;
53         }
54     }
55
56     /* set up structure for output color image */
57     /* Note that the type is 'c' rather than 'g' */
58     get_TIFF( &color_img, input_img.height, input_img.width, 'c' );
59
60     // declare and initialize integer to store the dimension of the point ➤
61     // spread function
62     int PSF_dim = 9;
63
64     // apply filter using applyFilter function as defined below main
65     applyFilter(&color_img, &input_img, filter, PSF_dim);
66
67     /* open color image file */
68     if ( ( fp = fopen( "color.tif" , "wb" ) ) == NULL ) {
69         fprintf( stderr, "cannot open file color.tif\n" );
70         exit( 1 );
71     }
72
73     /* write color image */
74     if ( write_TIFF( fp, &color_img ) ) {
75         fprintf( stderr, "error writing TIFF file %s\n", argv[2] );
76         exit( 1 );
77     }
78
79     /* close color image file */
80     fclose( fp );
81
82     /* de-allocate space which was used for the images */
83     free_TIFF( &(input_img) );
84     free_TIFF( &(color_img) );
85
86     return(0);
87 }
88
89 void error(char* name)
90 {
91     printf("usage:  %s  image.tif\n", name);
92     printf("this program reads in a 24-bit color TIFF image.\n");
93     printf("It then horizontally filters the green component, adds noise, ➤
94     \n");
95     printf("and writes out the result as an 8-bit image\n");
96     printf("with the name 'green.tif'.\n");
97 }
```

```
95     printf("It also generates an 8-bit color image,\n");
96     printf("that swaps red and green components from the input image");
97     exit(1);
98 }
99
100 // limitIntensity function definition
101 int limitIntensity(double inputValue) {
102     // declare an integer variable newValue and initialize it to zero
103     int newValue = 0;
104     // if input value parameter is less than zero, assign new value to 0
105     if (inputValue < 0) {
106         newValue = 0;
107     }
108     // if input value parameter is greater than 255, assign new value to 255
109     else if (inputValue > 255) {
110         newValue = 255;
111     }
112     // otherwise, assign new value to the input value parameter re-cast as an integer
113     else {
114         newValue = (int)inputValue;
115     }
116     return newValue;
117 }
118
119 // applyFilter function definition
120 void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, double** usedFilter, int PSF_dim) {
121     // declare and define N - the dimension of the point spread function (PSF)
122     int N = (PSF_dim - 1) / 2;
123     // declare and define image height and width based on input image TIFF struct methods
124     int img_height = input_img->height;
125     int img_width = input_img->width;
126     // declare doubles to store red, green, and blue value for each pixel
127     double redPlane, greenPlane, bluePlane;
128     // declare PSF variables
129     int m, n;
130     // declare variables to store current location within PSF
131     int a, b;
132     // for each pixel:
133     for (int i = 0; i < img_height; i++) {
134         for (int j = 0; j < img_width; j++) {
135             // initialize RGB values to zero
136             redPlane = 0.0;
137             greenPlane = 0.0;
138             bluePlane = 0.0;
```

```
139         // for each pixel in the PSF (9*9 in this case)
140         for (m = -N; m <= N; m++) {
141             for (n = -N; n <= N; n++) {
142                 // assign a and b to current PSF matrix location
143                 a = i - m;
144                 b = j - n;
145                 // if a and b are within the image boundaries
146                 if (a >= 0 && a < img_height && b >= 0 && b < img_width) {
147                     // apply filter by summing across PSF according to
148                     // difference equation for 2D filters
149                     redPlane += usedFilter[m + N][n + N] * input_img->color[0][a][b];
150                     greenPlane += usedFilter[m + N][n + N] * input_img->color[1][a][b];
151                     bluePlane += usedFilter[m + N][n + N] * input_img->color[2][a][b];
152                 }
153             }
154             // populate output image method for color after calling
155             // limitIntensity function to ensure acceptable RGB values
156             output_img->color[0][i][j] = limitIntensity(redPlane);
157             output_img->color[1][i][j] = limitIntensity(greenPlane);
158             output_img->color[2][i][j] = limitIntensity(bluePlane);
159         }
160     }
```


4. FIR Sharpening Filter

1. Derivation of $H(e^{j\mu}, e^{j\nu})$: *See Next Page*
2. Derivation of $G(e^{j\mu}, e^{j\nu})$: *See Next Page*

Let $h(m, n)$ be a low pass filter. For our purposes use

$$h(m, n) = \begin{cases} 1/25 & \text{for } |m| \leq 2 \text{ and } |n| \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

The unsharp mask filter is then given by

$$g(m, n) = \delta(m, n) + \lambda(\delta(m, n) - h(m, n))$$

where λ is a constant greater than zero.

Identity Used:

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$

Apply formula for DSFT & Simplify:

$$H(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m, n) e^{-j(\mu m + \nu n)}$$

$$= \sum_{n=-2}^2 \sum_{m=-2}^2 \frac{1}{25} e^{-j(\mu m + \nu n)}$$

$$= \frac{1}{25} \sum_{n=-2}^2 e^{-j\mu n} \sum_{m=-2}^2 e^{-j\nu m}$$

$$= \frac{1}{25} \left[(1 + e^{-2j\mu} + e^{2j\mu} + e^{-j\mu} + e^{j\mu}) (1 + e^{-2j\nu} + e^{2j\nu} + e^{-j\nu} + e^{j\nu}) \right]$$

$$= \frac{1}{25} \left[(1 + 2\cos(2\mu) + 2\cos(\mu)) (1 + 2\cos(2\nu) + 2\cos(\nu)) \right]$$

$$= \frac{1}{25} \left[\left(1 + 2 \sum_{k=1}^2 \cos(k\mu)\right) \left(1 + 2 \sum_{l=1}^2 \cos(l\nu)\right) \right]$$

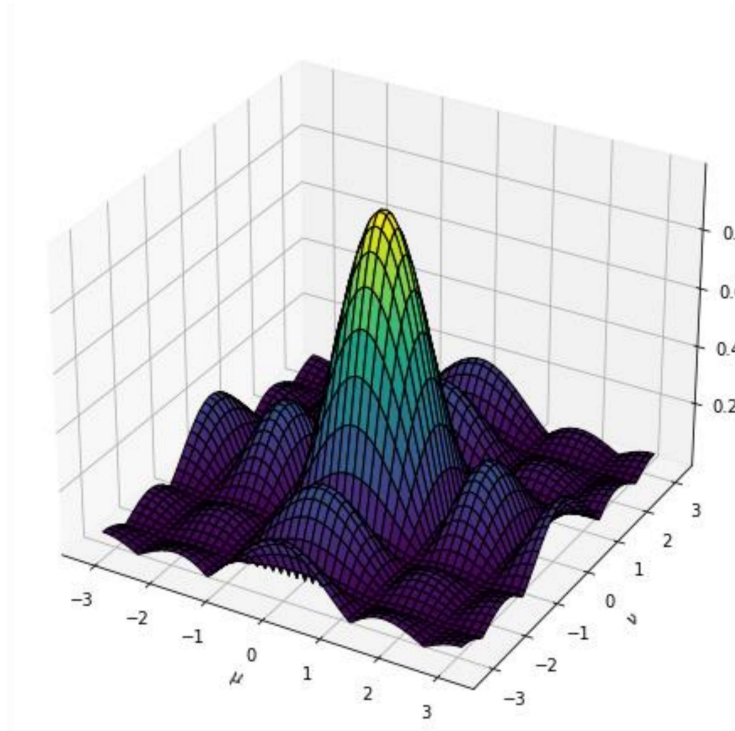
Apply formula for DSFT & Simplify, knowing $H(e^{j\mu}, e^{j\nu})$:

$$G(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} g(m, n) e^{-j(\mu m + \nu n)} = 1 + \lambda(1 - H(e^{j\mu}, e^{j\nu}))$$

$$= 1 + \lambda \left[1 - \frac{1}{25} \left[\left(1 + 2 \sum_{k=1}^2 \cos(k\mu)\right) \left(1 + 2 \sum_{l=1}^2 \cos(l\nu)\right) \right] \right]$$

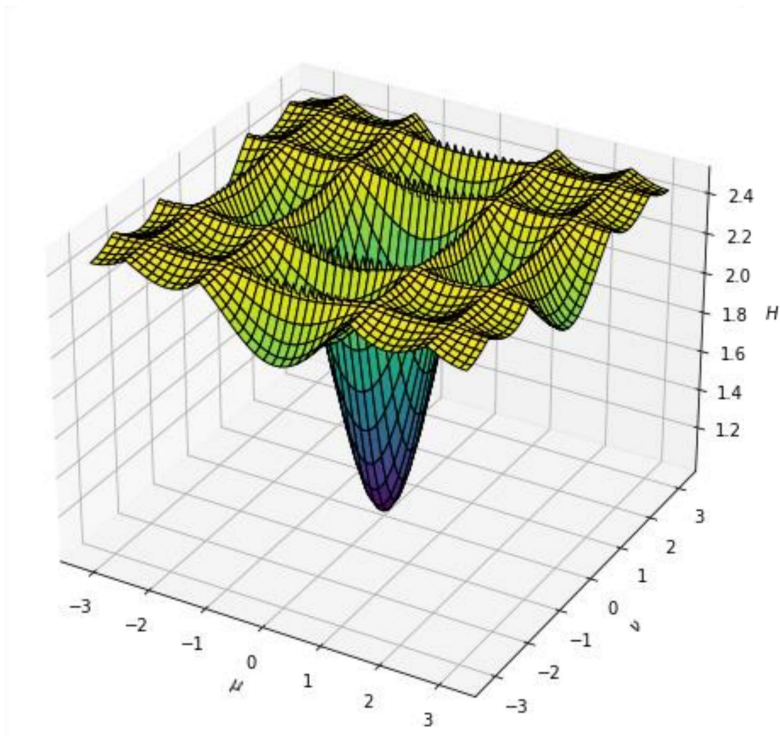
3. Plot of $|H(e^{j\mu}, e^{j\nu})|$

Magnitude of Frequency Response $|H(e^{j\mu}, e^{j\nu})|$ vs μ and ν



4. Plot of $|G(e^{j\mu}, e^{j\nu})|$ for $\lambda = 1.5$

$|G(e^{j\mu}, e^{j\nu})|$ vs μ and ν at $\lambda = 1.5$



5. Input color *imgblur.tif*



6. Output sharpened color image, *sharpened.tif*, for $\lambda = 1.5$



7. Listing of C Code: *See Next Page*

```
1
2 #include <math.h>
3 #include "tiff.h"
4 #include "allocate.h"
5 #include "randlib.h"
6 #include "typeutil.h"
7
8 void error(char *name);
9 // initialize limitIntensity function
10 int limitIntensity(double value);
11 // initialize applyFilter function
12 void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, ↗
    double** usedFilter, int PSF_dim, double lambda);
13
14 int main (int argc, char **argv)
15 {
16     FILE *fp;
17     struct TIFF_img input_img, color_img;
18
19     // adjustment: if number of arguments is greater than 3, since third ↗
    // will be lambda value
20     if ( argc > 3 ) error( argv[0] );
21
22     /* open image file */
23     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
24         fprintf ( stderr, "cannot open file %s\n", argv[1] );
25         exit ( 1 );
26     }
27
28     /* read image */
29     if ( read_TIFF ( fp, &input_img ) ) {
30         fprintf ( stderr, "error reading file %s\n", argv[1] );
31         exit ( 1 );
32     }
33
34     /* close image file */
35     fclose ( fp );
36
37     /* check the type of image data */
38     if ( input_img.TIFF_type != 'c' ) {
39         fprintf ( stderr, "error: image must be 24-bit color\n" );
40         exit ( 1 );
41     }
42
43     // declare 1D array of double pointers filter
44     double* filter[5];
45     // declare double to store scaling factor lambda - default value below
46     double lambda = 1.0;
47     if (argc > 2) {
```

```
48     lambda = atof(argv[2]);
49 }
50
51 // iterate through 1D array filter and allocate enough memory for 9  ➤
52 // doubles each
53 for (int i = 0; i < 5; i++) {
54     filter[i] = malloc(sizeof(double) * 9);
55 }
56 // populate filter array according to g(m,n)
57 for (int i = 0; i < 5; i++) {
58     for (int j = 0; j < 5; j++) {
59         if (i == 2 && j == 2) {
60             filter[i][j] = 1 + lambda * (1 - 1.0 / 25.0);
61         }
62         else {
63             filter[i][j] = lambda * (-1.0 / 25.0);
64         }
65     }
66 }
67
68 /* set up structure for output color image */
69 /* Note that the type is 'c' rather than 'g' */
70 get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
71
72 // declare and initialize integer to store the dimension of the point  ➤
73 // spread function
74 int PSF_dim = 5;
75
76 // apply filter using applyFilter function as defined below main
77 applyFilter(&color_img, &input_img, filter, PSF_dim, lambda);
78
79 /* open color image file */
80 if ( ( fp = fopen ( "sharpened.tif", "wb" ) ) == NULL ) {
81     fprintf ( stderr, "cannot open file color.tif\n");
82     exit ( 1 );
83 }
84
85 /* write color image */
86 if ( write_TIFF ( fp, &color_img ) ) {
87     fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
88     exit ( 1 );
89 }
90
91 /* close color image file */
92 fclose ( fp );
93
94 /* de-allocate space which was used for the images */
95 free_TIFF ( &(input_img) );
96 free_TIFF ( &(color_img) );
```

```
95
96     return(0);
97 }
98
99 void error(char *name)
100 {
101     printf("usage:  %s  image.tiff \n\n",name);
102     printf("this program reads in a 24-bit color TIFF image.\n");
103     printf("It then horizontally filters the green component, adds noise, \n");
104     printf("and writes out the result as an 8-bit image\n");
105     printf("with the name 'green.tiff'.\n");
106     printf("It also generates an 8-bit color image,\n");
107     printf("that swaps red and green components from the input image");
108     exit(1);
109 }
110
111 // limitIntensity function definition
112 int limitIntensity(double inputValue) {
113     // declare an integer variable newValue and initialize it to zero
114     int newValue = 0;
115     // if input value parameter is less than zero, assign new value to 0
116     if (inputValue < 0) {
117         newValue = 0;
118     }
119     // if input value parameter is greater than 255, assign new value to 255
120     else if(inputValue > 255) {
121         newValue = 255;
122     }
123     // otherwise, assign new value to the input value parameter re-cast as an integer
124     else {
125         newValue = (int)inputValue;
126     }
127     return newValue;
128 }
129
130 // applyFilter function definition
131 void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, double **usedFilter, int PSF_dim, double lambda) {
132     // declare and define N - the dimension of the point spread function (PSF)
133     int N = (PSF_dim - 1) / 2;
134     // declare and define image height and width based on input image TIFF struct methods
135     int img_height = input_img->height;
136     int img_width = input_img->width;
137     // declare doubles to store red, green, and blue value for each pixel
```

```
138     double redPlane, greenPlane, bluePlane;
139     // declare PSF variables
140     int m, n;
141     // declare variables to store current location within PSF
142     int a, b;
143     // for each pixel
144     for (int i = 0; i < img_height; i++) {
145         for (int j = 0; j < img_width; j++) {
146             // initialize RGB values to zero
147             redPlane = 0.0;
148             greenPlane = 0.0;
149             bluePlane = 0.0;
150             // for each pixel in the PSF (5*5 in this case)
151             for (m = -N; m <= N; m++) {
152                 for (n = -N; n <= N; n++) {
153                     // assign a and b to current PSF matrix location
154                     a = i - m;
155                     b = j - n;
156                     // if a and b are within the image boundaries
157                     if (a >= 0 && a < img_height && b >= 0 && b <
158                         img_width) {
159                         // apply filter by summing across PSF according to
160                         // difference equation for 2D filters
161                         redPlane += usedFilter[m + N][n + N] * input_img-
162                             >color[0][a][b];
163                         greenPlane += usedFilter[m + N][n + N] *
164                             input_img->color[1][a][b];
165                         bluePlane += usedFilter[m + N][n + N] * input_img-
166                             >color[2][a][b];
167                     }
168                 }
169             }
170             // populate output image method for color after calling
171             // limitIntensity function to ensure acceptable RGB values
172             output_img->color[0][i][j] = limitIntensity(redPlane);
173             output_img->color[1][i][j] = limitIntensity(greenPlane);
174             output_img->color[2][i][j] = limitIntensity(bluePlane);
175         }
176     }
177 }
```


5. IIR Filter

1. Derivation of $H(e^{j\mu}, e^{j\nu})$: *See Next Page*

Let $h(m, n)$ be the impulse response of an IIR filter with corresponding difference equation

$$y(m, n) = 0.01x(m, n) + 0.9(y(m-1, n) + y(m, n-1)) - 0.81y(m-1, n-1)$$

where $x(m, n)$ is the input and $y(m, n)$ is the output.

Jack Girard
ECE637
Lab 1 Work

Identity Used:

$$e^{i\phi} = \cos \phi + i \sin \phi$$

Taking Z-Transformation of difference equation:

$$Y(z_1, z_2) = 0.01X(z_1, z_2) + 0.9z_1^{-1}Y(z_1, z_2) + 0.9z_2^{-1}Y(z_1, z_2) - 0.81z_1^{-1}z_2^{-1}Y(z_1, z_2)$$

$$Y(z_1, z_2) - 0.9z_1^{-1}Y(z_1, z_2) - 0.9z_2^{-1}Y(z_1, z_2) + 0.81z_1^{-1}z_2^{-1}Y(z_1, z_2) = 0.01X(z_1, z_2)$$

$$Y(z_1, z_2)(1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}) = 0.01X(z_1, z_2)$$

$$H(z_1, z_2) = \frac{Y(z_1, z_2)}{X(z_1, z_2)} = \frac{0.01}{1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}}$$

$$H(e^{j\mu}, e^{j\nu}) = \frac{0.01}{1 - 0.9e^{j\mu} - 0.9e^{j\nu} + 0.81e^{-j\mu}e^{-j\nu}}$$

$$H(e^{j\mu}, e^{j\nu}) = \frac{0.01}{1 - 0.9e^{j\mu} - 0.9e^{j\nu} + 0.81e^{-j(\mu+\nu)}}$$

Further Simplifying $H(e^{j\mu}, e^{j\nu})$ and calculating $|H(e^{j\mu}, e^{j\nu})|$

$$= \frac{0.01}{1 - 0.9(\cos \mu - j \sin \mu) - 0.9(\cos \nu - j \sin \nu) + 0.81(\cos(\mu + \nu) - j \sin(\mu + \nu))}$$

$$= \frac{0.01}{[1 - 0.9 \cos \mu - 0.9 \cos \nu + 0.81 \cos(\mu + \nu)] + [0.9 j \sin \mu + 0.9 j \sin \nu - 0.81 j \sin(\mu + \nu)]}$$

$$= \frac{0.01}{\underbrace{[1 - 0.9(\cos \mu + \cos \nu) + 0.81 \cos(\mu + \nu)]}_{=a} + j \underbrace{[0.9(\sin \mu + \sin \nu) - 0.81 \sin(\mu + \nu)]}_{=b}}$$

$$= \frac{0.01}{a + j b} = \frac{0.01}{a + j b} \cdot \frac{a - j b}{a - j b} = \frac{0.01(a - j b)}{a^2 + b^2} = k(a - j b) ; k = \frac{0.01}{a^2 + b^2}$$

$$|k(a - j b)| = \sqrt{(ka)^2 + (kb)^2} = k\sqrt{a^2 + b^2} ; \begin{aligned} a &= 1 - 0.9(\cos \mu + \cos \nu) + 0.81 \cos(\mu + \nu) \\ b &= 0.9(\sin \mu + \sin \nu) - 0.81 \sin(\mu + \nu) \end{aligned}$$